HW#3: Cache Optimization

朱致伶 Chu Chih Ling, 110550062

***Abstract*—This report analyzes the performance of 2, 4, 8-way set associated cache in FIFO / LRU replacement policy respectively. FIFO has more stable performance compared with LRU. For FIFO policy, 4-way associated cache has the best performance. As for LRU policy, it has low cache count, but comes with higher cache latency and lower CoreMark score compared with FIFO.**

I. Introduction

In this homework, we first analyze the current 4-way set-associative cache count and latency in following aspects: read/write, hit/miss. Then, modify the current model to 2-way, 8-way set associated cache and the cache replacement policy, compare the performance between these models.

II. Statistics and Analysis

Most of the signals and code modifications happen in dcache.v in synthesis. Setting the following environment variables to default values: DITERATION=0, DTOTAL_DATA_SIZE=4000. Calculate the read/write and hit/miss cache counts and latency, then analyze the different N-way set associative cache in FIFO/LRU replacement policy in following aspects:

A. *Read/Write and Hit/Miss Cache*

The p_strobe_i signal sends the processor request signal of data cache, which triggers the cache operation. To depart read cache from write cache, I catch the p_rw_i and rw signal. As for departing the hit cache from miss cache, I catch the cache_hit signal. By the way, the definition of cache latency is the number of cycles between p_strobe_i and p_ready_o signals, which doesn't include the p_ready_o cycle.

The tricky part for catching signals is the one-cycle latency between the cache_hit and other input signals. To tackle this problem, I create two registers to record the state of strobe and ready, which are both one-clock cycle after p_strobe_i and p_rw_i respectively. Signals can be divide to two sets for better understanding, p_strobe_i and p_rw_i belong to the first set, and p_strobe, p_ready, rw, cache_hit belong to the second set. The signals in the second set is one-clock after the ones in the first set.

From Table 1, we can see the cache related statics of 4-way associated cache with FIFO policy. We have following conclusions: (1) The average cache latency of hit and miss cache are 1 and 50.33 cycles respectively, which meets the assumption that miss cache needs more cycles to get data from main memory. (2) Since the high rate of cache hit, the hit cache cycles dominate the average cache cycles, which is 1.12 cycles per cache. (3) The ratio of read and write cache is approximately 4:1, implies that CoreMark implements more read operations than write in cache.

TABLE I. Statics of Categories of Cache

Categories of Cache	***Total Cache Count***	***Cache Rate***	***Total Cache Latency***	***Average Cache Latency***
Total Cache	90,979,607	100%	100,712,047	1.107 cycle
Hit Cache	90,782,305	99.78%	90,782,305	1.0 cycle
Miss Cache	197,302	0.22%	9,929,742	50.33 cycle
Read Cache	72,309,496	79.47%	80,477,627	1.11 cycle
Write Cache	18,670,111	20.52%	20,234,420	1.08 cycle

TABLE II. Statistics of Different Combination Cache

Categories of Cache	***Total Cache Count***	***Cache Rate***	***Total Cache Latency***	***Average Cache Latency***
Total Cache	90,979,607	100%	100,712,047	1.107 cycle
Read Hit Cache	72,143,981	79.30%	72,143,981	1.0 cycle
Write Hit Cache	18,638,324	20.49%	18,638,324	1.0 cycle
Read Miss Cache	165,515	0.18%	8,333,646	50.35 cycle
Write Miss Cache	31,787	0.035%	1,596,096	50.21 cycle

Table 2 is just a supplement to know the detailed distribution of different combinations of cache.

B. *N-ways associated Cache (2-way, 4-way, 8-way)*

Theoretically, as the N increases, the associativity rises, causing the miss rate to decrease. Also, since the associated cache searches all entries in a given set at once, one comparator is needed for N entries. Therefore, less comparators is needed for larger N.

From Table 3, we get the following conclusions: (1) 4-way set associated cache has the best performance among the 3 caches. (2) 8-way cache has hit/miss rate, average cache latency and CoreMark score value close to the ones of 4-way cache. However, it has miss cache latency slightly higher than the one of 4-way. (3) As for the 2-way cache, the lowest hit rate and high miss cache latency, cause the highest average cache latency among the 3 cases. Also, it has the lowest CoreMark score.

I speculate the reason behind the highest miss rate of 2-way cache is the small amount of stored data in each way compared to the others. Making the processor frequently access the main memory to get and switch data. As for the reason behind the highest miss cache latency of 8-way cache, the large data size to be moved from main memory to cache, and the data searching policy may also affect the latency.

TABLE III. STATISTICS OF N-WAY ASSOCIATED CACHE IN FIFO

N-Way Cache	***Hit Cache Rate***	***Miss Cache Rate***	***Average Miss Cache Latency***	***Average Cache Latency***	***CoreMark Score***
2-way	99.69%	0.31%	50.3499 cycle	1.1521 cycle	24.7577
4-way	99.78%	0.22%	50.3276 cycle	1.1070 cycle	24.9056
8-way	99.78%	0.22%	50.3510 cycle	1.1074 cycle	24.9045

C. Least Recently Used (LRU) Replacement Policy

In addition to FIFO replacement policy, another common cache replacement policy is least recently used. As the name, the policy chooses the one unused for the longest time and replaced it in the cache set.

To implement the LRU policy, I only come up with the most intuitive way, declaring a two-dimensional array to record the accumulated hit count of each way for all cache lines, and replace the one with least hit count when cache miss. Code below is the main part of LRU policy I implemented:

```
if (S == RdfromMemFinish) begin
        LRU_table[line_index][hit_index] <= LRU_table[line_index][hit_index] + 1;
        idx2 = 0;
        for (idx = 1; idx < N_WAYS; idx = idx + 1) begin
                if(LRU_table[line_index][idx] < LRU_table[line_index][idx2])
                        idx2 = idx;
        end
        LRU_cnt[line_index] <= idx2;
end
```

From Table 4 and 5, we observe the following results: (1) As the N increases in LRU policy, the total cache count and coremark score decreases in such a large scale, and the average cache latency increases. (2) Compared to 4- and 8-way cache of the two replacement policy, 2-way LRU cache performance varies the least from FIFO. (3) As N increases, the miss rate rises significantly, however, the average miss cache latency only decreases slightly.

TABLE IV. STATISTICS OF N-WAY ASSOCIATED CACHE IN LRU

N-Way Cache	***Total Cache Count***	***Total Cache Latency***	***Average Cache Latency***	***CoreMark Score***
2-way	90,979,603	155,276,894	1.7067 cycle	23.70864
4-way	68,793,481	505,273,143	7.3448 cycle	13.62945
8-way	26,635,468	392,537,662	14.7374 cycle	8.86915

TABLE V. STATISTICS OF N-WAY ASSOCIATED CACHE IN LRU

N-Way Cache	***Hit Cache Rate***	***Miss Cache Rate***	***Miss Cache Count***	***Miss Cache Latency***	***Average Miss Cache Latency***
2-way	98.57%	1.43%	1,301,961	65,599,252	50.3849 cycle
4-way	87.11%	12.89%	8,867,147	445,346,809	50.2241 cycle
8-way	72.08%	27.92%	7,435,562	373,337,756	50.2097 cycle

The following are some discussions of LRU policy results from above:

(1) Compared to the statistics between FIFO and LRU, LRU has lower cache count, processor doesn't need to access memory frequently to fulfill read/write requests, which is good to the system performance. However, LRU has low cache hit rate compared with FIFO, triggering data replacement in high rate. It is a trade-off between total cache count and cache hit rate. Neither of the two policies, LRU or FIFO, outperforms each other in all aspect, each of them has their own pros and cons.

(2) Compared to different N-way cache of LRU, as the N increases exponentially, the total cache count and cache hit rate drops sharply, the average cache latency increases, while average miss cache latency slightly decreases. We can see that LRU replacement policy doesn't have stable performance as N increases. It is a nice try to analyze the performance of LRU in different perspectives.

(3) The LRU coremark score is significantly lower than FIFO. As N increases, the score dramatically decreases. The reason behind is the increase sizes of the 2-dimensional array (N_LINE x N_WAYS), which account for a large area of the board, leading to low performance.

III. DISCUSSION

Below are some topics I would like to have a deeper discussion about:

A. Improvement of LRU Replacement Policy

The LRU approach I implemented requests a great amount of for loop and large sizes registers. For loop in verilog is parallel tasks running simultaneously, which is inefficient. Due to the low coremark score of LRU replacement policy, I have a discussion with my classmates for efficient and simple method of LRU. Instead of storing the accumulated hit count of each way per line, we can store the order of last hit way per line.

The method only requests 8 bits per entry, each 2 bits represent one way, in which the value ranges from 0 to 3. The left side of the 8 bits are the most frequently used way, the right side are the least frequently used one. Compared to the method I implemented, it only requests log(N_WAYS) x N_LINE bits to store the state and decide the victim cache select.

B. Choosing N for N-way Associated Cache

It is obvious that 4-way cache has the best performance among the three cases. However, I was a little stuck in choosing the N with best performance for the LRU policy. It is a trade-off between total cache count, hit rate, latency and coremark score, which is worth a comprehensive analysis as future work.

IV. SUMMARY

This homework mostly analyzes the cache statistics, I tried hard to come up with a efficient way to implement the LRU policy, but end up failed. However, it is great to have chances get to know cache mechanism better.